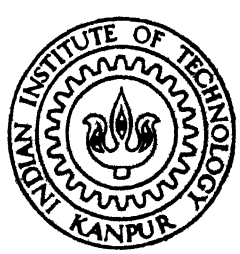
A Methodology for Synthesis of DSP Architectures

by

S. ARVIND



Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

February 1997

A Methodology for Synthesis of DSP Architectures

A Thesis Submitted

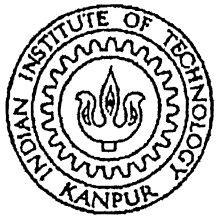
in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

S. ARVIND



Department of Electrical Engineering
Indian Institute of Technology
Kanpur - 208016, UP, India

February 1997

Dedicated

To

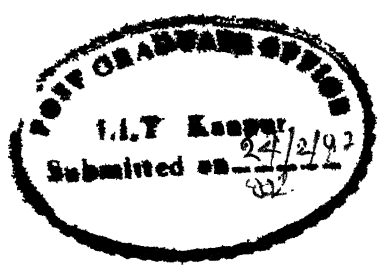
Amma & Appa

EE-1997-M-ARV-MET

Certificate

This is to certify that the work contained in the thesis entitled A Methodology for Synthesis of DSP Architecture by S. Arvind has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Sumana Gupta,
Associate Professor,
Department of Electrical Engineering,
Indian Institute of Technology, Kanpur.



Acknowledgement

I would like to express my sincere gratitude to Dr. Sumana Gupta for her ever available counsel and guidance. She has been a constant source of inspiration and motivation during all phases of this work.

I express my sincere gratitude towards Cadence India for awarding me the Cadence Fellowship for the thesis work. I take this opportunity to express my sincere thanks to Dr. S. K. Roy who motivated me in accepting the Fellowship.

My sincere thanks to Kbala for sparing his time in teachin me some of the fundas of Graph Theory. I thank Gomes and our C Master (Sh)Anan for their help and cooperation during the course of the work. My sincere thanks to Dixitji, Neeraj, Purwar, Snsimhan, Muthup for making my stay a memorable one. Thanks to the fun loving Ghat Mandali for their ever available help, esp. Desai, Gokhale. Bhole etc and for teaching me Marathi. Thanks to the members of Tamil Mandram who took the pain of correcting my Tamil. Special thanks to those known or unknown who helped me and made my stay here a beautiful and memorable experience at IITK.

Abstract

The high level synthesis of dedicated architectures has become a crucial step in the design process. The present work tries to generate various architectural specifications for a dedicated system executing DSP algorithm. The crucial steps towards synthesis are scheduling of operations and allocation of hardware resources. Attempt has been made to generate specifications of the proposed architecture which has a minimum interconnection cost and reduced number of hardware resources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in VLSI technology have dramatically reduced the cost of all forms of information processing. This effect is more pronounced in the field of real time signal processing. The continuous flow of data together with the complexity of the algorithms impose severe computational demands that often cannot be met by general purpose(programmable) processors. The stringent requirement of high computational speed are often met by dedicated system architectures which exploit the potential concurrency existing within the algorithm. The programmable processors are costly both in terms of area and power dissipation.

High level synthesis of dedicated architectures for real time DSP systems has received considerable attention in recent times [1] [2][3][4]. It has become a crucial step in the design flow because of many new applications which require high sampling rates. Sample rates depend upon the application, ranging from few KHz. for speech systems to tens and hundreds of MHz. for real time image and radar processors. Therefore some algorithms may require thousands of computational steps to be performed for each signal datum, leading to com-

putation demands exceeding billions of arithmetic operations per second.

The complete synthesis process as it exists today encompasses the following major steps

1. **Behavioral description**

   The algorithm is expressed in some high level language which can be easily translated into a signal flow graph. In the graph based representation each node of the graph represents some operation to be performed and the edges represent the data path.

2. **Operation scheduling**

   Operation scheduling is the process of the determining the assignment of operation to time slots of synchronous system (each operation corresponds each node in the signal flow graph). Each operation is subject to data flow dependencies, which ensures that the input operands for the scheduled operation are available when the operation is scheduled.

3. **Hardware allocation**

   The last step towards the synthesis is the allocation and binding. Besides allocating the nodes/operation onto the specific resource/functional units, the storage units are assigned to share intermediate values. Also the specification of the control circuits needs to be generated.

Among the above steps the scheduling and hardware allocation are the two most difficult and crucial steps in the synthesis and hence maximum efforts are being put towards the same.

In recent times many synthesis systems have been developed for automated design of high performance dedicated architectures especially for DSP applications[1].

CATHEDRAL[5] implements the integer programming model for microcode scheduling to solve small problems. For larger problems a heuristic list scheduler is used. SPAID[6] utilizes a linear programming model to solve the scheduling task. ALPC[2] utilizes a linear programming model for the scheduling task. Most of these scheduler require excessive CPU time for large problems. Many of these are critical path schedulers and are capable of producing schedule if the iteration bound is greater or equal to the critical path. MARS[1] has used a new concurrent scheduling and resource allocation algorithm based on iterative loops. The resource allocation is done by a technique called folding [1]. However the final architecture developed has complex interconnections. Madisetti [3] has developed a new scheduling algorithm for multiprocessor environment.

1.1 Problem statement

Synthesis of VLSI architectures from a DSP algorithm behavioral description has been addressed using a number of different approaches as given in [3][7]. In the present work an attempt has been made to get an optimal schedule and to synthesize a hardware with minimal interconnection cost. The interconnection of various functional units which are bound after scheduling tends to increase the area while routing. Moreover the interconnect delays also tends to degrade the performance of the system(the interconnect delays being proportional to the length of the wire). Although the different techniques that exist have been reported separately by various authors, we attempt to combine certain aspects of reported work to get a better design.

1.2 Outline of the thesis

Chapter 2 gives a brief description of some of the existing basic scheduling methodologies. Chapter 3 discusses the scheduling methodology adopted. The details of the scheduler along with an algorithm used for scheduling has been described. Chapter 4 discusses the hardware allocation methodology adopted for the proposed target architecture. The results obtained for different flow graphs are discussed in Chapter 5. Finally Chapter 6 concludes the work presented in the thesis and outlines the scope of future work.

Chapter 2

Existing Methodology

The goal of a synthesizer is to convert the behavioral description of the system into a description of an architecture that can implement the required behavior. The description of the system can be done using a high level language such as C. The first step towards the process is to convert this description into a dependence graph(signal flow graph); all subsequent steps use this graph. Typically a set of constraints namely timing and/or resource constraints are specified for the same.

2.1 Scheduling

The process of scheduling determines the area-time(speed) trade-off and attempts to schedule the operations optimally. If the system is subject to speed (timing)constraints the scheduling methodology tries to parallelize the operations to meet the timing constraints. Conversely, if there is a limit on the number of available resources the scheduler tries to schedule onto the specified resources only [7].

In general three types of scheduling problems exist. They are:

1. *Time constrained Scheduling*: The scheduler tries to schedule the operations into a fixed number of time steps using minimum number of resources

2. *Resource Constrained Scheduling*: The scheduler tries to find the fastest schedule given the constraints on the number of resources.

3. *Feasible Constrained Scheduling*: Given the constraints on the number of resources and the maximum time steps, the scheduler tries to find out the optimal schedule satisfying both these constraints.

The Scheduling problem has been proved to be an NP-Complete problem [7]. and hence only heuristic algorithms are possible.Use of heuristic algorithm does not guarantee the optimality of the solution. A good heuristic algorithm must have a low time complexity and should also produce an optimal or near optimal solution for a large set of practical problems.

2.2 Review of the existing basic techniques

1. **As Soon As Possible (ASAP) Heuristic**
   This is one of the simplest technique. The operations on the data flow graph are scheduled step by step from the first control step to the last. An operation is called ready operation if all its predecessors are scheduled. This procedure repeatedly schedules the ready operations to the next time step until all the operations are scheduled.

2. **As Late As Possible (ALAP) Heuristic**
   This type of scheduling follows a very similar procedure as the ASAP

except that it starts from the last time step and moves towards the first. In this case an operation is scheduled if all the successors are scheduled

Both ASAP and ALAP are generally followed by an Integer Linear Programming (ILP) formulation to the problem. As ASAP gives the earliest possible time and ALAP gives the latest possible time for each operations. These algorithms in some sense give the lower and upper bound on the scheduling time. The ILP uses these informations to minimize the cost of resources. Such technique is more suited for time constrained scheduling, since ALAP requires the last time step to be specified.

3. **Freedom Based Scheduling Heuristic** [3].

   In this scheduling methodology the critical nodes are scheduled first. The nodes that are not critical are assigned one at a time according to their degree of freedom. The degree of freedom is calculated as the difference between the ASAP time and the ALAP time (calculated with reference to the scheduled nodes).

4. **Force Directed Scheduling Heuristic** [8].

   The "force" values are calculated for all operations at all feasible time steps. The pairing of operations and time step that has the most attractive force is selected and assigned. After assignment, the force values of the unscheduled operations are reevaluated. This process of assignment and evaluation is iterated until all operations are assigned.

Both freedom based and force directed scheduling are 'global' not only in the way they select the next operation to be scheduled but also in the way they decide the control steps.

There is always a trade off between the two constraints namely timing and the resource constraints. The constraints are generally given by the user and in some cases dictated by the algorithm itself. Time Constrained scheduling is important for designs targeted towards applications in real time systems. For a DSP system the sample rate of the input data stream dictates the maximum time allowed for executing an algorithm on the present data sample before the next sample arrives. Since the sample rate is fixed, the main objective is to minimize the cost of the hardware. Given the control step length, the sample rate can be expressed in terms of number of control steps that are required for executing a DSP algorithm. The Scheduling algorithm also has to be tailored to suit the different target architectures.

2.3 Hardware Allocation

The process of allocating scheduled nodes onto a hardware and designing a control unit is the next task. Scheduling assigns operations to control steps and thus converts a behavioral description into a set of register transfers that can be described by a state transition table. The datapath needs to be derived from the register transfers assigned to each step. The datapath in the present model is a netlist composed of three types of components namely functional units, storage units and interconnection units.

Functional units such as Adders, Multipliers etc. execute the operations specified in the behavioral description. Storage units such as registers, register files, ROM etc. hold the values of the variables generated and consumed during execution of the behavior. The interconnection units such as buses and multiplexers transfer data between the functional units and storage units.

The data path allocation consists of two essential tasks:

1. Unit selection
2. Unit binding

Unit selection process determines the number and types of components to be used in the design. This is in some sense dictated by the behavioral description itself. Unit binding involves the mapping of variable and operations in the scheduled data flow graph into functional, storage and interconnection units, while ensuring that the design behavior operates correctly on the selected set of components. For every operation in the data flow graph a functional unit is needed that is capable of executing the operation. For each variable that is used across several time steps in the scheduled data flow graph (DFG), a storage unit is needed to hold the data during the variable's lifetime. Besides the design constraints imposed on the original behavioral description, additional constraints on the binding process are imposed by the target architecture. For example a functional unit can execute only one operation in any given time step, the number of multiple accesses to storage unit during a control step is limited to the number of parallel ports on the unit etc.

The interconnection topology that supports data transfers between the storage and functional units is one of the factors that has a significant influence on the data path performance. The complexity of the interconnection topology is defined as the maximum number of the units between two ports of functional or storage units. The complexity is also defined in terms of the number of communication paths and how complex they are.

The basic requirement of the unit selection process is that the number of

units performing a certain type of operations must be equal to or greater than the maximum number of operations of that type to be used in any time step. The objective of interconnection binding is to minimize the sharing of interconnection units and thus the interconnection cost while still supporting the conflict free data transfer.

Chapter 3

Scheduling Approach Adopted

3.1 Behavioral description

To extract the maximal parallelism from an algorithm it is necessary to describe the algorithm in a form that preserves its basic computational and structural properties. The computational properties of an algorithm are described by the set of operations and associated computational delays. The structural properties are a result of data and control dependencies. The absence of dependencies indicate the possibility of simultaneous computations.To capture these properties the algorithm is specified in the form of a graph.

Different forms of graphs can be classified as:[3]

1. **Shift Invariant flow graphs**
   In a Shift invariant flow graph, the graph operations and the structure does not change with time. For linear systems a shift in the input results in the shift of the output. This is shown in the Figure 3.1.

2. **Generic Flow Graph**

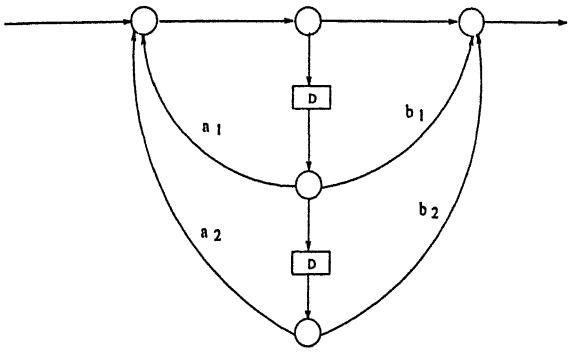
A generic flow graph is a directed flow graph of processing nodes communication edges and ideal delays. A node as denoted in Figure 3.1 represents an arithmetic or logical function that is performed with a fixed computational delay. An edge represents a dependence relation between processing nodes. An ideal delay Z^{-1} (D) partitions the graph into separate iterations. The delay represents memory registers and stores past values of intermediate calculations.

3. **Fully Specified Flow graphs(FSFG)**

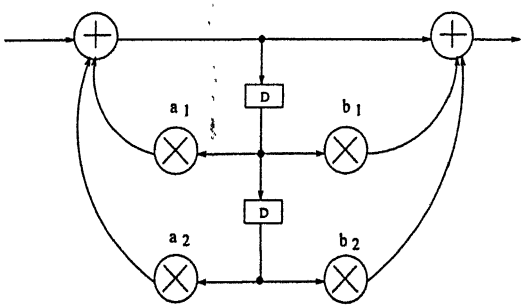
   To exploit maximum parallelism it is necessary to specify the nodes with smallest granularity possible. A FSFG is a generic SFG in which node operations are additionally constrained to be atomic operations of the constituent processors(functional units) on which the algorithm is to be implemented. A FSFG representation is shown in Figure 3.1.

All the three representation for a second order IIR filter are shown in Figure 3.1.

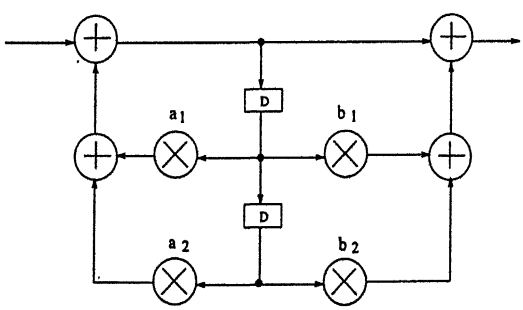
In the present methodology the DSP algorithm is behaviorally described using a fully specified SFG which is a finite edge-weighted and node-weighted directed graph denoted by $G(N, E)$, where N is the set of nodes. Each node corresponds to an operation in the SFG(algorithm). The node weight $n(d)$ corresponds to the operation execution delay specified by the user according to a targeted technology and available library. The set of edges is denoted by E where each edge e is a triplet (u, v, w). Each edge e represents an unidirectional communication path between nodes u and $v(u, v \in N)$ and w is a non-negative number signifying the delay in the edge e. This delay represents the number of registers placed along the edge in the SFG and is used to



Shift Invariant Flow Graph



Generic Flow Graph



Fully Specified Flow Graph

Figure 3.1: Different Forms of Signal Flow Graphs

represent Z^{-1} (the delay element). The edge represents data interconnection and determines precedence relations between data. A single delay represents a phase shift of one sample. A node can be activated only if there are enough samples on all the incoming edges.

For the sake of simplicity the following restrictions are imposed on the SFG [9].

1. The graph is computable i.e., no delay free directed loops are present. This condition alleviates the possibility of having problems with races, oscillation and asynchronous latches.

2. The graph is finite. Hence terminating bounded iterations can unfold into a finite graph.

3. Conditional and data dependent executions are not present. This condition has been used since DSP algorithms are generally data independent.

4. Only a single throughput is allowed for all the SFG. Thus for multiple processing, the different sampling periods are taken as multiples of a constant rate common to every operation in the flow graph. The use of such flow graph facilitates the application of structural transformation besides being a suitable representation for DSP algorithms. Such conditions easily extend this methodology to 2D and multirate systems.

3.2 Scheduling representation

The Scheduler has to give a schedule for executing a particular algorithm on the targeted architecture such that the tasks are executed with greatest

efficiency in the specified time. The factors that affect the efficiency of such an architecture are

1. Operator Precedence.
2. Number of Functional Units(FU).
3. Iteration Period (latency between the iterations).
4. Maximmum Throughput Delay.
5. Interconnection cost.

Most of the available literature[1] focusses on reducing the number of functional units to a lower bound and not much attention has been given towards interconnection cost. The wiring (connections) between various functional units and the storage units takes up towards in the final design. If the interconnections are too complex, then inspite of reducing the number of units the design will be very inefficient in terms of area. Moreover the communication delay may also add to the inefficiency of the design. In the present methodology a simplification of the interconnections has been attempted. Other factors such as throughput delay and FU utilizations has also been considered. The scheduler is designed to takes care of all these constraints. The schedule is represented in the form of a matrix. The schedule matrix is a graphical representation of the processor(functional unit) vs time that portrays one period of the schedule in implementing the FSFG.Each entry in the matrix corresponds to a particular node of the graph. The memory access time by a particular unit can be (for simplicity) added to the computational delay of each unit. The number of columns in the matrix equals the iteration period. The subscript in each entry represents the iteration index of each operation in the FSFG.

For example, a $-n$ subscript corresponds to an operation that has occured n iterations before while a $+n$ represents an operation that will occur after n iterations.

Example is given by a schedule matrix for a second order IIR filter is as shown in Figure 5.4 *Processors* $\downarrow$

$$\begin{bmatrix} 1_0 & 2_0 & 5_1 \\ 5_0 & 8_0 & 7_0 \\ 3_0 & 4_1 & 4_1 \\ 6_1 & 6_1 & 3_1 \end{bmatrix}$$

$\rightarrow$ *timeslots* $\rightarrow$

The matrix also indicates the iteration index for each such execution.The actual time index of the processor schedule is related to the time slot of the schedule matrix as

$$t_{act} = t_{SM} - T_0 \times i$$

where t_{act} is the actual time index of operation in the schedule, T_0 is the iteration period bound and i is the iteration index of the operation

As there is always a tradeoff between area and time (speed), a scheduler generating an optimal schedule is prefered. In general, depending on the initial graph there maybe a lower bound on the iteration period. This lower bound on the time period is dictated by the loops present in the FSFG. The optimality criteria that needs to be satisfied while assuming unconstrained interconnections are given as follows:

Rate Optimality

The measure of rate optimality is the *iteration period bound (IPB)*. The iteration period bound represents the minimum available latency between

the iterations of the given flow provided sufficient processing elements are available.The IPB is given as

$$IPB = max\left\lceil \frac{\sum_{\forall j} d_j}{n_l} \right\rceil,$$

where $\sum_{\forall j} d_j$ is the sum of computational throughput delay of the loop l with number of ideal delays as n_l.

The schedule that achieves this bound is rate optimal. The rate optimal schedule produces outputs at the fastest average rate. A critical loop is thal loop whose loop bound equals the IPB. In general all other loops can be executed in a much lesser time and thus the residue time dictates the scheduling flexibility of the constituent nodes.

Delay Optimality

Periodic throughput delay is the measure of Delay optimality. The *Periodic Delay Bound (PDB)* represents the minimum average throughput delay from input to output at a given iteration period. The schedule that operates at PDB has the shortest delay between the input and output and is delay optimal. The PDB is given as

$$PDB = max\left[\sum_{\forall j} d_j - n \times IPB \right],$$

where n is the number of ideal delays along a particular path.

Processor Optimality

This estimates the cost of resources used in performing the computation of the algorithm. The lower bound on the number of processor used while meeting the rate optimality is called the *Processor Bound* and is given as

$$PB = max\left\lceil \frac{\sum_{\forall j} d_j}{T} \right\rceil,$$

where $\sum_{\forall j} d_j$ is the sum of computational delay of all nodes and T is the rate at which the processors are operating and it equals IPB. However it can also be defined in terms of functional units and can be defined as

$$PB_U = max \left\lceil \frac{\sum_{\forall j} d_j}{T} \right\rceil, \forall j \in U,$$

where PB_U is the lower bound on the number of functional units of type U.

3.3 Presynthesis Transformation

One of the parameters that degrades the performance of a system is the throughput delay. The minimum achievable throughput delay equals the iteration period for reasons stated before. In order to reduce the throughput delay to the lower bound and to pipeline the operations, the data flow graph can be transformed through some of the available graph transformations[11][12] such as .

1. Unfolding Transformation
2. Retiming Transformation

3.3.1 Unfolding Transformation

Unfolding[12] with a factor N is a transformation in which the transformed graph explicitly describes the presedence constraints in N consecutive iterations. For multiprocessor scheduling the unfolding transformation can always produce a graph which can be scheduled rate optimally. The optimal unfolding factor N is the least common multiple of all loop delay counts.Unfolding

increases the granularity of the representation and thus additional concurrency is achieved. However the disadvantage of this is that N iterations needs to be considered for scheduling and hence the representation becomes complex.

3.3.2 Retiming Transformation

The process of retiming involves moving around the delays in the DFG such that the total number of delays in any loop remains unaltered and the I/O behaviour of the system is unaffected.

Retiming was first proposed by Leiserson[11] to reduce the clock period of pipelined systems. Retiming preserves the loopbound while it does change the throughput delay of the systems to its iteration period. It also pipelines the operations by reducing the critical path. It changes intra iteration precedence constraints to inter iteration precedence constraints and vice-versa.

Removal of a fixed number of delays from each of the incoming edge of any node and addition of the same number of delays to each of the outgoing edge of the same node is one such example of retiming applied locally to the node. This is equivalent to cutset transformation around the node. However this transformation has been tailored for DSP algorithm in [13][4].

Retiming is a transformation $R : G \longrightarrow G_r$ of the original graph $G = \{V, E(n)\}$ to the retimed graph. $G_r = \{V, E(n_r)\}$ where the new delay $n_r(e)$ of the edge $e(u, v)$ connecting nodes u and v is defined by the equation

$$n_r(e) = n(e) + r(u) - r(v),$$

where $r(v) : V \longrightarrow Z$ transforms the node to an integer value The retiming is valid as long as $n_r(e) \geq 0$. As the distribution of delays is needed according

to some linear criterion, this problem has been formulated as shortest path formation. The method followed is briefly described below.

1. Reduce the edge weights n(e) by 1 along the edges which does not have a delay in the original DFG.

2. Use Bellman Ford algorithm[10] to find the shortest path from all nodes to the output. $r(v)$ equals the weight of the shortest path so computed from each node.

3. Compute the new edge delays using the information of $r(v)$ along all paths.

The limitation of this method is that Bellman Ford algorithm cannot be applied if the sum of the edge weights in a loop is negative after the applying step 1.

3.4 Description of the Scheduler

The scheduling methodology that has been adopted for the present work is the feasible constrained scheduling. The advantages of this approach are as follows

1. Time required to find a solution by feasible scheduling is less than that found by other scheduling techniques since only a part of the solution space needs to be searched for optimization.

2. The maximum number of functional units and the maximum number of time steps can be easily estimated for any complex graph.[14]

3. This approach also allows use of an expert system to control the area-time trade off.

As indicated earlier the scheduler has to determine the fastest schedule that meets the specified constraints. Using a selection of different resource sets leads to several scheduling solutions from which it is possible to determine the number of resource/performance tradeoff for the target system. DSP algorithms are relatively free of conditional statements and branching i. e. the nodes of the SFG represents some arithmetic operations (generally additions or multiplications) only and hence are relatively simple to schedule as compared to a very general algorithm.

The scheduling algorithm used is briefly described in the following section

3.4.1 Description of the algorithm

Initially the scheduler enumerates all the recursive (loops) and non recursive (I/O paths) sections of the specified graph (input DSP algorithm). These sections of the graphs are used in computing the optimality criteria namely the *iteration bound, periodic delay bound* and *processor bound.* The I/O paths have been identified by a depth first search of the edges incident from the input nodes.

Once all the loops/paths have been identified and the IPB and PDB has been computed the critical loops and critical paths are marked. As defined earlier a critical loop has its loop bound equal to the iteration bound while the critical path has its throughput delay as the maximum. On applying the retiming transformation the critical path gets reduced and operations get pipelined. This reduces the maximum throughput delay to the iteration bound. This

process does not affect the schedule but it does have an effect on the hardware allocation part explained later. The methodology applied for scheduling is the Freedom Based Scheduling. As explained earlier this scheduling technique exploits the flexibility of each node and schedules accordingly. The Scheduler schedules the nodes in order of decreasing constraints. Therefore the highly constrained nodes are the critical nodes and hence are scheduled first. Once the critical path/loops are identified, the constituents nodes are marked critical. The critical nodes have zero flexibility and needs to be scheduled first. All other nodes have some flexibility while scheduling, the flexibility being dictated by the corresponding loop or path slack time. This slack time is the difference between the loop/path bound and the corresponding maximum bound.

Some of the assumptions used to simplify the scheduling process are

1. System is fully synchronous.
2. Computational delay of all operations are data independent.
3. Data access time i.e. either zero or can be added to the corresponding functional unit computational delay.

The algorithm used for the generation of an optimal schedule is being described subsequently. Care has been taken in scheduling the adjacent operations onto the same processor. It also ensures a minimal change in the states of the control circuitry and a least amount of communication between the processors is required. If the obtained schedule is implemented on a multiprocessor environment, it would be very costly in terms of area and programming efforts.For a full custom design of application specific ICs (for signal processing) such schedules would be very inefficient as hardware allocation is to be done sep-

arately. The hardware allocation allocates the individual operations onto the corresponding functional units.

```
            result=Schedule();
        }

}
Schedule()
{
        node=ChooseNxtNode();

        if(node==critical)
        {
            if(LookforSlot(node)==noSlot)
            return(invalid);
            else
            copyNode(node);
        }

        else
        ComputeStartTime(node);
        result=invalid;

        while(result==invalid)
        {
            if(LookforSlot(node)==noSlot)
            return(invalid);
            else
            copyNode(node);
```

```
            result==Schedule();

            if(result==valid)
            return(valid);
            else
            AdjustTimeSlot();
        }
}
ComputeStartTime(node)
{
```

This function computes the earliest and the latest time(start times) between which the node could be scheduled.The function first chooses a scheduled node adjacent to the node and through the precedence constraints computes the start times. depending upon the selection of the scheduled node either earliest or the latest time is given the priority and is first tested as the beginTime.

```
}
AdjustTimeSlot()
{
```

This function tries to adjust the beginTime within the flexible limit of the node and then continues testing the other node.

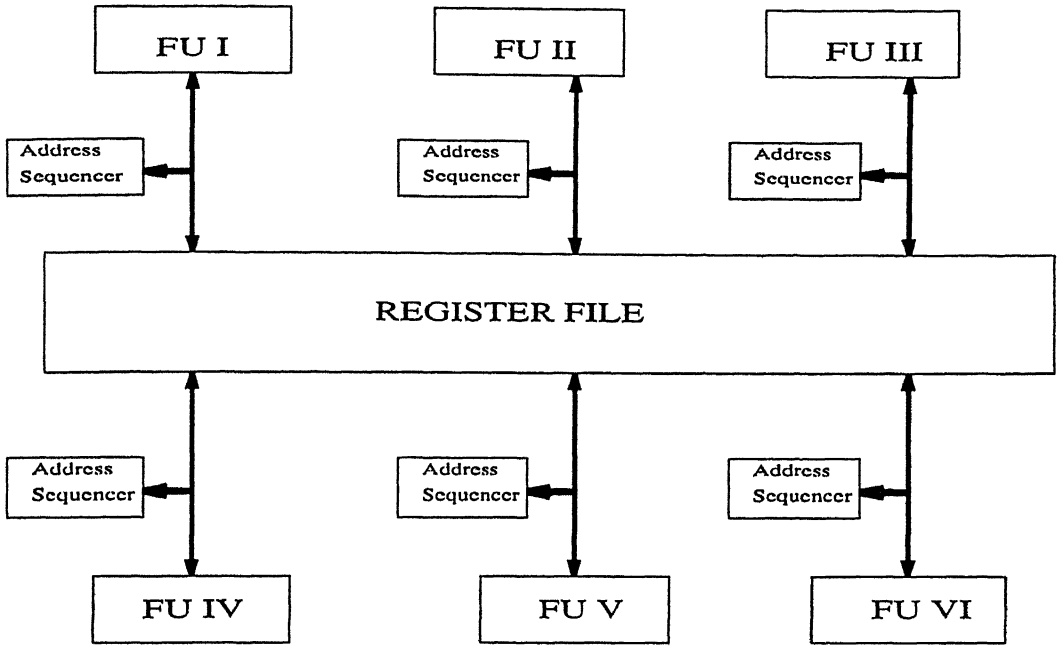
```
}
```

Chapter 4

Hardware Allocation

The last step in the synthesis process is the resource allocation part. The resource allocation process allocates various scheduled operations onto the functional units i. e. the scheduled flow graph is mapped onto the available hardware blocks. The mapping transforms the scheduled graph into two sub-graphs namely the datapath structure graph and the controller statemachine graphs. The datapath structure graphs gives the mapping of node operations onto the various functional units and the mapping of intermediate variables onto the registers (register file). The controller state machine graph gives the state transitions of various state-machine controllers which enables proper read/write operations by the functional units. As mentioned earlier most of the available literature focusses on the reduction of the processing elements assuming unconstrained interconnections. The complex and criss cross connections create problems while routing and causes interference which degrade the performance of the system. This besides increasing the area also increases the communication delay. Thus in order to reduce interconnections a general architecture targeted for our purpose is being proposed. The operations scheduled are being mapped onto this target architecture.

4.1 Target architecture



TARGET ARCHTECTURE

Figure 4.1: Target Architecture

The general Architecture onto which the synthesizer is aimed at is as shown in the Figure 4.1. The main components of the architecture are

1. Functional Units such as adders/multipliers.
2. Register file
3. Various Control Units

Functional Units

The type of functional units such as adders, multipliers etc. are dictated by the type of operation/nodes present in the dataflow graph. The minimum number of units of each type is determined by finding the maximum number of concurrently scheduled nodes. Various node operations of the same type are mapped onto the same functional units.

Register Files

Various intermediate variables generated during the iterations are held in a memory. This memory has been implemented as a register file. The number of registers present in this file is dictated by the methodology adopted for mapping various variables. The specifications of the register file such as number of ports are generated by the program. The number of ports present in the file is dictated by the maximum number of read and write operations onto the memory at any particular time. Since the whole synthesis process is aimed towards a full custom design, the register file and other units are not directly picked up from a semicustom library. Hence it is being assumed that a register file with any number of ports can be easily designed. These I/O ports can also be multiplexed depending on the usage of the number of ports at a particular instant of time and thus they can be binded in the final design.

State Machines

Various functional units read and write from various memory locations.The addresses of these locations are generated by the various state machines which

constitute the control circuitry. The state transition table of these machines are being generated as part of the specifications .

The advantage of such an architecture is that the interconnections between various units are few as well as simple. The criss cross connections are also being avoided. This in turn reduces the area and the intercommunication delay thereby improving the efficiency.

4.2 Mapping Procedure

The mapping procedure of the operations onto the various resources(functional Units) has to follow from the operation scheduling only. As described earlier, both scheduling and allocation tasks are interrelated and in some sense on doing one the other gets done automatically. The schedule matrix obtained so for gives a mapping procedure of the nodes onto a multiprocessor environment with unconstrained intercommunication links.

However for a full custom design of a real time signal processing application such mapping is inefficient both in terms of area and power. Thus it would be better to separate out various functional units out of these and map operations onto them separately. In order to achieve a lower bound on the number of resources, it is necessary to share the resources. Resource sharing is an assignment of a resource to more than one operation. The primary goal of this is to reduce the area of a circuit by allowing multiple number of concurrent operations to share the same hardware operator. The operations that would share the same resource is constrained to be of same type.

Let the number of variables that needs to be bound onto the same resource be

N. These N variables can be assigned to the same physical resource if and only if each pair of these N variables do not have usage conflicts. Let each variable be represented as a node in an undirected graph known as compatibility graph. The sharability of variables can be represented as an edge between the corresponding nodes in the graph. Consider a Graph G consisting of a finite number of nodes and a set of undirected edges containing pairs of nodes. A nonempty collection C of nodes present in G forms a complete graph if each node in C is connected to every other node of C. We define a Clique as a complete Graph C with respect to G if C is not contained in any other complete graph contained in G[9]. Thus mapping of elements (both nodes and node variables) reduces to a partitioning problem.

Consider the graph $G = (V, E)$ to be partitioned, where $V = \{v_1, v_2...v_N\}$ is the set of nodes and $E = \{e_1, e_2...e_m\}$ the set of edges. The graph has to be partitioned into K partitions such that each partition forms a complete subgraph(i.e a Clique) subject to the following conditions

1. Each partitions V_k for $k \leq K$ contains sets of compatible nodes i.e. nodes which are of same type and are scheduled non concurrently.

2. Minimum number of partitions: This constraint is required to minimize the cost of design as more number of partitions would mean more number of units and a corresponding increase in area and interconnection cost. However the number of partitions cannot be reduced below a certain number since there is a lower bound on each functional units as discussed earlier. Hence the number of partitions K is subjected to the constraint

   $Kmin \leq K \leq Kmax$

   where Kmax is the maximum number of possible partitions and is directly

related to the worst case design performance.

3. The number of interconnections in each level of partitions have to be minimized.

The required objective functions are:

- min. K
- min. $\sum_{i=1}^{K} \left| m_i - \frac{N}{K} \right|$
- min. $\sum_{i=1}^{K} w_i$

where N is the number of nodes in the graph G, m_i is the number of nodes in the i^{th} cluster and w_i are the corresponding weights.These weight are measure of the profit of combining nodes and hence would determine the overall cost. Some or all of these constraints are applied to the original scheduled graph while allocating the hardware.

The partitioning problem can be formulated as a weighted clique partitioning problem. The original undirected graph which is the compatibility graph is constructed and weights are attatched to each of the edges.

4.2.1 Functional Unit Allocation

Each type of functional unit is allocated independently. For each type of unit a separate compatibility graph is formed. The number of nodes in this undirected graph equals the number of nodes of the same type in the original data flow graph.All such nodes are the member of the compatibility graph. An edge is between two nodes iff both these nodes occur non concurrently in the

schedule. A weight is attatched to each of these edges which is a measure of interconnection cost. The weight w of an edge $e(u, v, w)$ is computed by the following method.

1. If nodes u,v occur concurrently then the edge does not exist.
2. If the nodes u,v are scheduled non concurrently then the corresponding edge weight equals the minimum number of nodes the data computed by node u needs to traverse before reaching node v.

The constructed graph is partitioned by the weighted clique partitioning method described later.

4.2.2 Register Allocation

The process of register allocation is similar to the one suggested for functional unit binding process except for the method of constructing the compatibility graph. As described the architecture is such that each functional unit reads and writes onto a memory location in a register file. Therefore the register allocation process needs to reduce the number of memory locations. The output of each node in the data flow graph are the variables written onto the memory location and hence need to be minimized. The nodes of the compatibility graph are the node outputs. A variable is said to be alive from the time it is generated till the time of its last use. Similarly the variable is dead from the time of its last use to the time of its next generation. The start time and the end time can be easily computed from the schedule matrix.Two variables can be combined if the live periods of the two variables do not overlap. The live and dead status of all the variables can be represented as a matrix refered to

as lifetime matrix. A compatibility graph can be constructed from the matrix in the following way

1. A complete graph consisting of all nodes is first created
2. From the lifetime matrix for the pair of nodes which have overlapping lifetimes, the corresponding edge is removed.
3. The weights are added to the remaining edges depending upon the way the nodes are allocated to the functional units.
    - If both the nodes are allocated onto the same functional unit then weight W =100
    - If both the nodes are of the same type and are allocated to different functional unit then weight W =50
    - If both the nodes are of different type then weight W =10

The constructed graph is partitioned by the weighted clique partitioning method The algorithm used for the partitioning is briefly described.

Clique Partitioning Algorithm

```
Clique(OldGraph,OldClusters)
{
        CreateEdgList(OldGraph,EdgeLists);
        result=unsuccessful;
        Edge=pop(EdgeLists);
        while(result==unsuccessful)
        {
             FindDisjointEdgs(EdgeList,Edge,indx,NewList);
             NoOfClusters=CreateClusters(NewList,NewClusters,OldClusters);
             RecreateGraph(NewGraph,NewList);
             if(NoOfClusters==MinReqd)
             return(success);
            else
            if(NewGraph=NotEmpty)
            result=Clique(NewGraph,NewClusters);
            indx=indx+1;
        }
}
FindDisjointEdges(EdgeList,Edge,indx,NewList)
{
        This function finds the edges which are non touching. The variable
indx keeps track of the combinations already tried (in case of back tracking).
}
```

Chapter 5

Results and Discussion

The suggested methodology has been tried out on various commonly know signal processing flow graphs such as

1. Lattice filter
2. IIR filter
3. FIR filter
4. Wave digital filter

All these graphs were retimed and the synthesised architecture was tested by behaviorally describing the same. The details of the architecture specifications are being discussed. Most of the DSP algorithm uses two types of operations namely multiplication and addition. For the sake of simplicity and comparison it is being assumed that the multiplier takes twice the time taken by an adder. It is also being assumed that the delay of the address decoding and the control circuitry is negligible as compared to the execution times of the functional units.

5.1 Lattice Filter

5.1.1 Scheduling

A second order Gray Markel Lattice filter structure as shown in the Figure 5.1 has been used for synthesising.

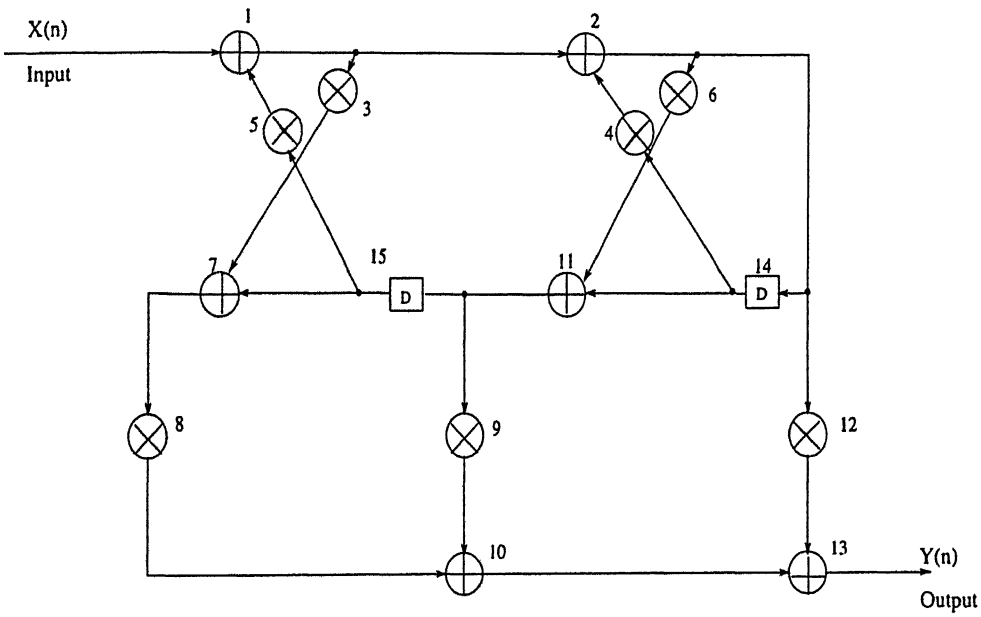


Figure 5.1: Second Order Lattice Filter

This flow graph has 3 loops and 6 I/O paths. Under the assumptions stated earlier the various bounds for this flow graph are

1. Iteration Bound = 7
2. Processor Bound =3 with 1 adder and 2 multiplier

Using these bounds, the schedule obtained by the scheduler is given by

$$\begin{bmatrix} 1_0 & 2_0 & 6_0 & 6_0 & 11_0 & 9_0 & 9_0 \\ 10_{-1} & 13_{-1} & 12_0 & 12_0 & & 3_1 & 3_1 \\ 4_0 & 5_0 & 5_0 & 7_0 & 8_0 & 8_0 & 4_1 \end{bmatrix}$$

The actual time of execution of an operation is related to the time slot in the matrix as

$t_{act} = t_{SM} - T_0 \times i,$

where $t_{act}$is the actual time index of operation in the schedule, T_0 is the iteration period bound and i is the iteration index of the operation.

The various delays present in the circuit is taken care of by the iteration index as discussed in Section 3.2 .

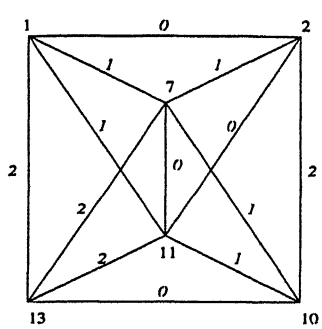
5.1.2 Resource allocation:

As discussed earlier the resource allocation process consists of two steps namely FU allocation and register allocation. This process not only maps the scheduled graph onto the available hardware resources, but also fixes the state transitions of the state machines used as control circuitry.

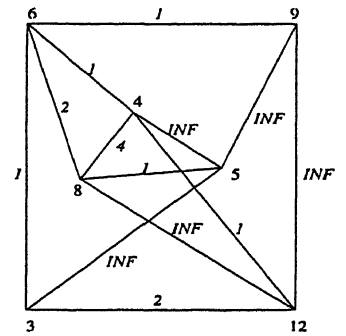
FU allocation

The number of adders required for this circuit is two as there are two addition operations scheduled concurrently in the schedule. Similarly the number of multipliers required is four .The compatibility graph can be constructed using the information on the concurrency provided by the schedule matrix. The graph so obtained for both adders and multipliers are shown in Figure 5.2.

This graph is subjected to a minimum clique partitioning algorithm. The



Compatibility Graph for Adder



Compatibility Graph for Multiplier

Figure 5.2: Compatibility Graphs for Lattice Filter

partitioning method takes care of the profit measure while forming clusters The nodes binded onto various FUs are listed in the Table 5.1. The details of the scheduling and allocation scheme is described in the Appendix.

Functional Unit	*Operation Allocated*			
Adder 1	Nodes	1	2	7
	Time	0	1	3
Adder 2	Nodes	10	13	11
	Time	0	1	4
Multiplier 1	Nodes	3	6	
	Time	5	2	
Multiplier 2	Nodes	4	5	8
	Time	6	1	4
Multiplier 3	Nodes	9	12	
	Time	5	2	

Table 5.1: FU - Node Mappings

Register allocation

As indicated earlier the memory required is being used in the form of a register file. The register allocation process gives the details of the location from where a particular FU reads and writes and at the time at which the transfer occurs. The lifetime chart formed is shown in Table 5.2.

node 1	L	D	D	D	D	D	D
node 2	D	L	L	L	L	L	D
node 3	D	D	D	D	D	D	L
node 4	L	D	D	L	L	L	L
node 5	D	D	L	D	D	D	D
node 6	D	D	D	L	D	D	D
node 7	D	D	D	L	D	D	D
node 8	D	D	D	D	D	L	L
node 9	D	D	D	D	D	D	L
node 10	L	D	D	D	D	D	D
node 11	L	D	D	D	L	L	L
node 12	L	D	D	D	D	D	D
node 13	L	L	L	L	L	L	L

Table 5.2: Life Time Chart for Lattice Filter

The compatibility graph obtained from this chart is subjected to weighted clique partitioning problem. The weights are so adjusted that the transitions of the state in various state machines are minimum, thus simplifying its design. The read and write location of various units is shown in Figure 5.3.
As is evident from the figure, the number of ports of the register file required is 3 since maximum of 3 read and write operations takes place at a particular instant of time.

The synthesised circuit has a throughput delay of 9 which is greater than the

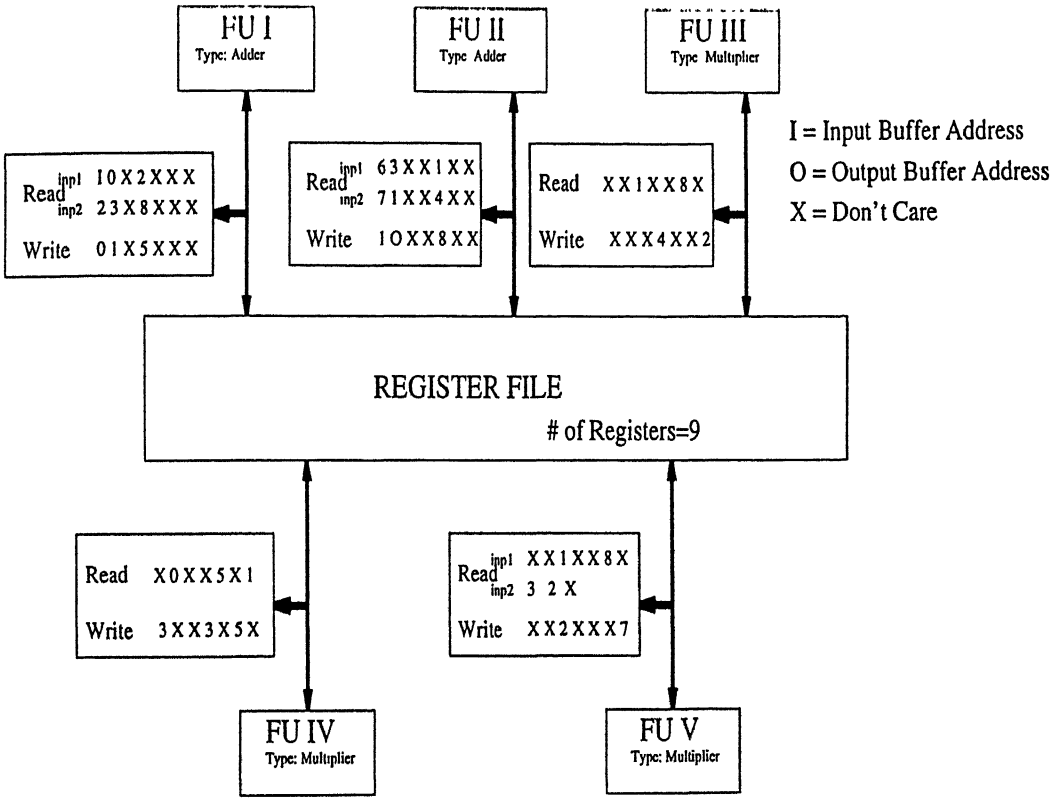


Figure 5.3: Synthesised Architecture for Lattice filter

iteration period. This can be reduced by retiming. However by the algorithm suggested in Section 3.3.2 the retiming is not possible due to the presence of negative cycles[10].

5.2 IIR fIlter

5.2.1 Scheduling

A Second order IIR filter is shown in Figure 5.4. The number of loops and I/O paths in this flow graph are 2 and 3 respectively. The various bounds of this DFG are

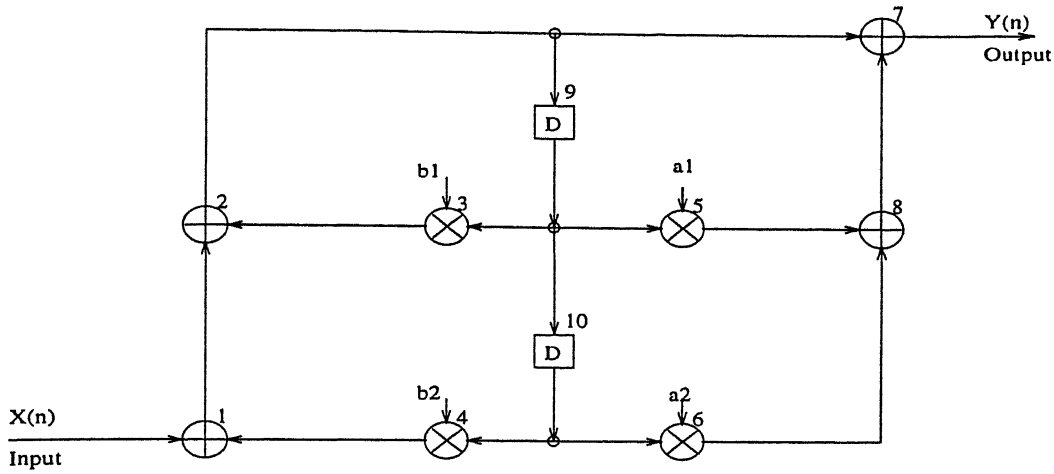


Figure 5.4: Second Order IIR Filter

1. Iteration Bound = 3

2. Processor Bound =4 with 2 adder and 3 multiplier

The shown graph does not have negative cycles and hence retiming could be performed. The retimed circuit used for synthesis is shown in Figure 5.5. On

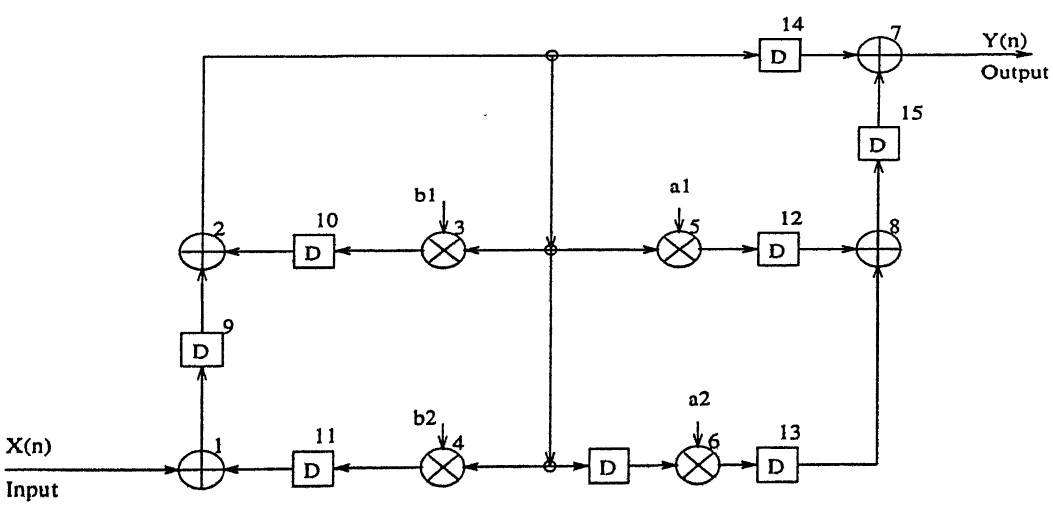


Figure 5.5: Retimed IIR Filter

applying the Scheduling algorithm the schedule so obtained is shown below.

$$\begin{bmatrix} 1_0 & 2_0 & 5_1 \\ 5_0 & 8_0 & 7_0 \\ 3_0 & 4_1 & 4_1 \\ 6_1 & 6_1 & 3_1 \end{bmatrix}$$

5.2.2 Resource Allocation

FU Allocation

The number of adders required for this graph is 2 since there are 2 additions operations scheduled concurrently. Similarly the number of multipliers required are 4. The nodes binded to various functional units are as listed in Table 5.3.

Functional Unit	*Operation Allocated*
Adder 1	Nodes 1 2
	Time 0 1
Adder 2	Nodes 8 7
	Time 1 2
Multiplier 1	Nodes 3
	Time 2
Multiplier 2	Nodes 4
	Time 0
Multiplier 3	Nodes 5
	Time 2
Multiplier 4	Nodes 6
	Time 1

Table 5.3: FU - Node Mapping

Register Alloaction

The lifetime chart of various variables in the scheduled graph of Figure 5.5 is shown in Table 5.4. The minimum number of register needed equals 5.

node 1	L	D	D
node 2	L	L	L
node 3	L	D	D
node 4	D	L	L
node 5	L	D	D
node 6	L	D	L
node 7	L	D	L
node 8	D	L	D

Table 5.4: Lifetime Chart for IIR filter

The architecture details are shown in the Figure 5.6

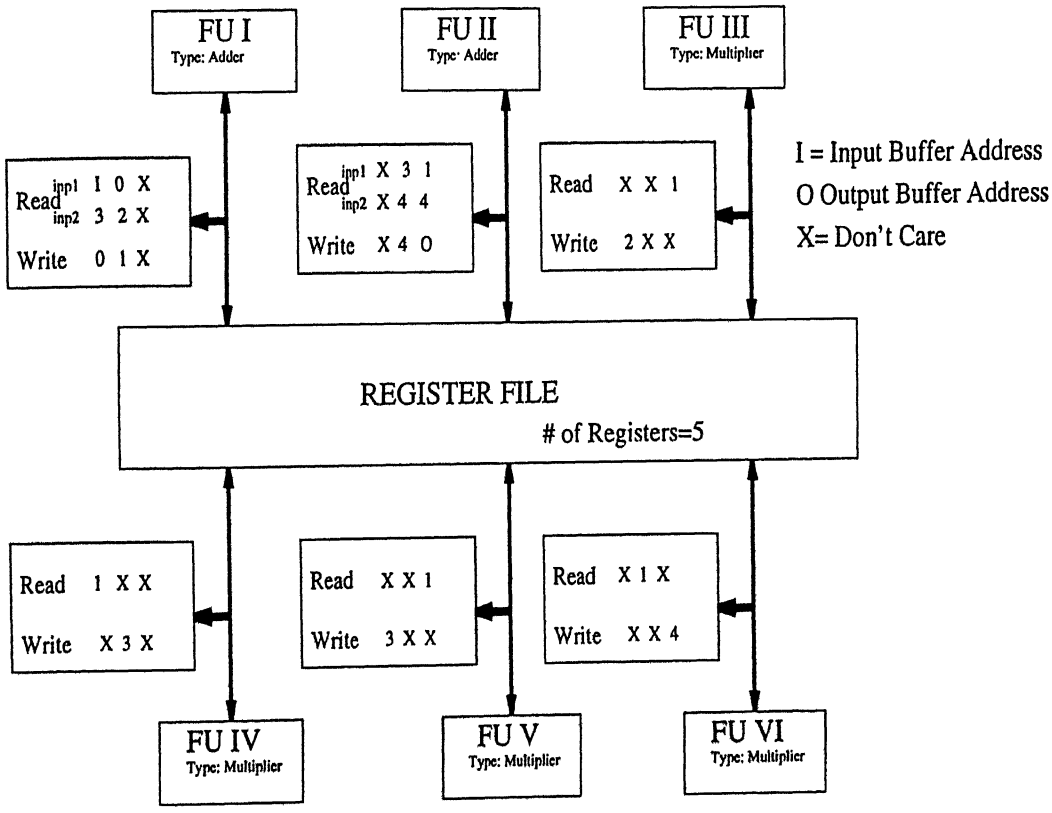


Figure 5.6: Synthesised Architecture of IIR filter

5.3 FIR fIlter

5.3.1 Scheduling

A Fifth order FIR filter is shown in Figure 5.7. The number of loops and I/O

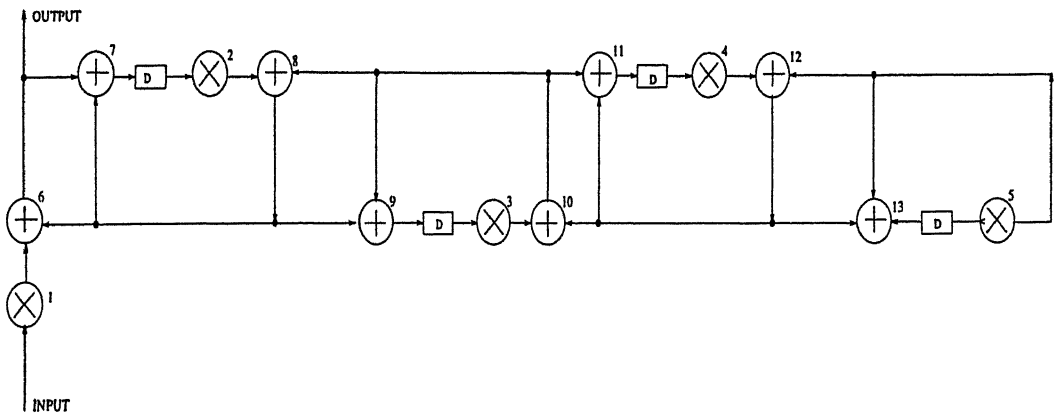


Figure 5.7: Fourth Order FIR Filter

paths in this flow graph are 8 and 1 respectively. The various bounds of this DFG are

1. Iteration Bound = 4
2. Processor Bound =4 with 2 adder and 2 multiplier

The shown flow graph has negative cycles and hence retiming cannot be performed. Hence the flow graph shown in Figure 5.7 is synthesised directly. On applying the Scheduling algorithm the schedule so obtained is shown below.

$$\begin{bmatrix} 1_0 & 1_0 & 6_0 & 7_0 & 2_1 \\ 2_0 & 8_0 & 9_0 & 3_1 & 3_1 \\ 10_1 & 11_1 & 4_2 & 4_2 & 12_2 \\ 13_1 & 5_1 & 5_1 & & \end{bmatrix}$$

5.3.2 Resource Allocation

FU Allocation

The number of adders required for this graph is 3 since there are 3 additions operations scheduled concurrently. Similarly the number of multipliers required are 3. The nodes binded to various functional units are listed in Table 5.5.

Functional Unit	*Operation Allocated*				
Adder 1	Nodes	6	7	10	12
	Time	2	3	0	4
Adder 2	Nodes	8	9		
	Time	1	2		
Adder 3	Nodes	11	13		
	Time	1	0		
Multiplier 1	Nodes	1	3		
	Time	0	3		
Multiplier 2	Nodes	2	5		
	Time	4	1		
Multiplier 3	Nodes	4			
	Time	2			

Table 5.5: FU - Node Mapping

Register Alloaction

The lifetime chart of various variables in the scheduled graph is shown in the Table 5.6. The minimum number of register needed equals 6.

The architecture details are shown in the Figure 5.8.

node 1	D	L	D	D	D
node 2	L	D	D	D	D
node 3	D	D	D	D	L
node 4	D	D	D	L	D
node 5	D	D	L	L	L
node 6	L	D	L	L	L
node 7	D	D	D	L	D
node 8	D	L	L	D	D
node 9	D	D	L	D	D
node 10	L	L	D	D	D
node 11	D	L	D	D	D
node 12	L	D	D	D	L
node 13	L	D	D	D	D

Table 5.6: Lifetime Chart for FIR filter

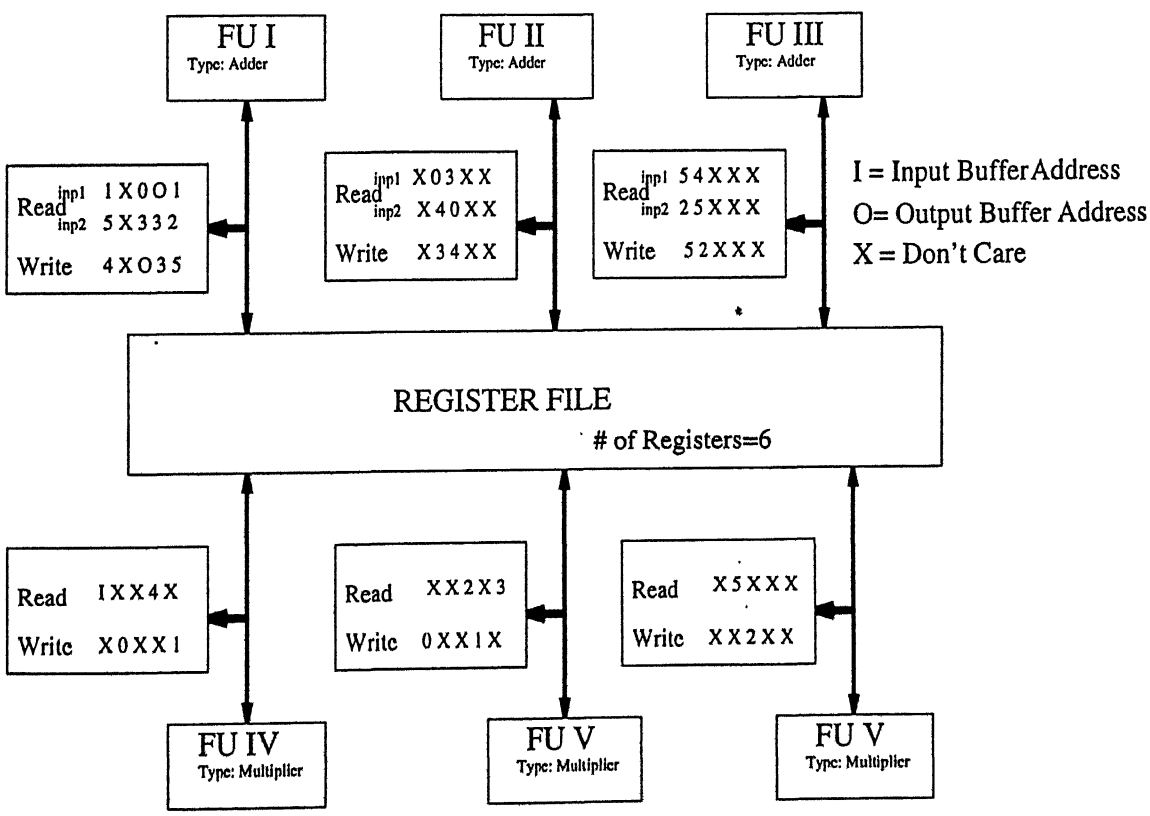


Figure 5.8: Synthesised Architecture of FIR filter

5.4 Wave Digital fIlter

5.4.1 Scheduling

A Fifth order Wave Digital filter is shown in Figure 5.9. It may be noted that this filter structure has been used as a benchmark circuit in the literature[] and has 45 loops. The various bounds of this DFG are

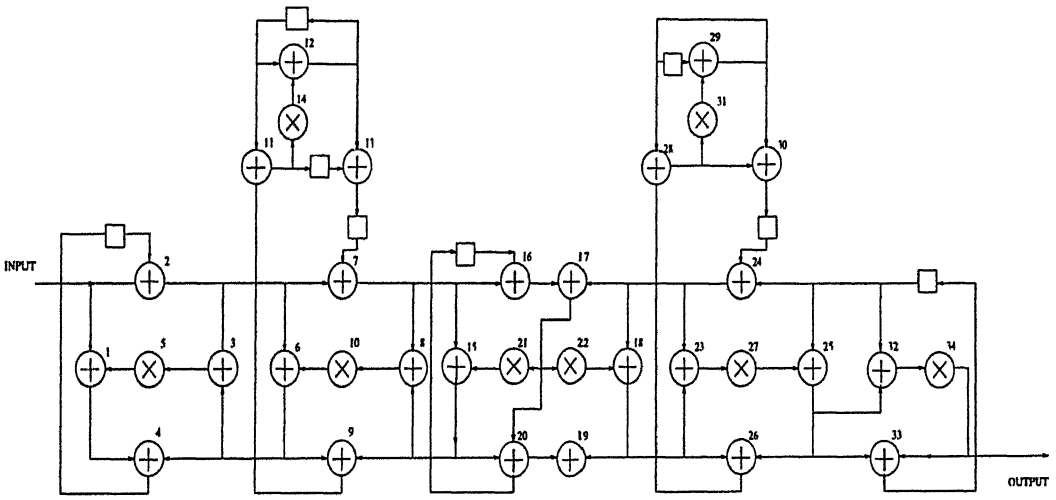


Figure 5.9: Fifth Order Wave Digital Filter

1. Iteration Bound = 16
2. Processor Bound = 4 with 3 adder and 2 multiplier

The retimed circuit is shown in Figure 5.10. However for the sake of comparison with existing methodologies [1] only the results of unretimed graph has been discussed. On applying the Scheduling algorithm to the graph in Figure 5.9 the schedule so obtained is as shown .

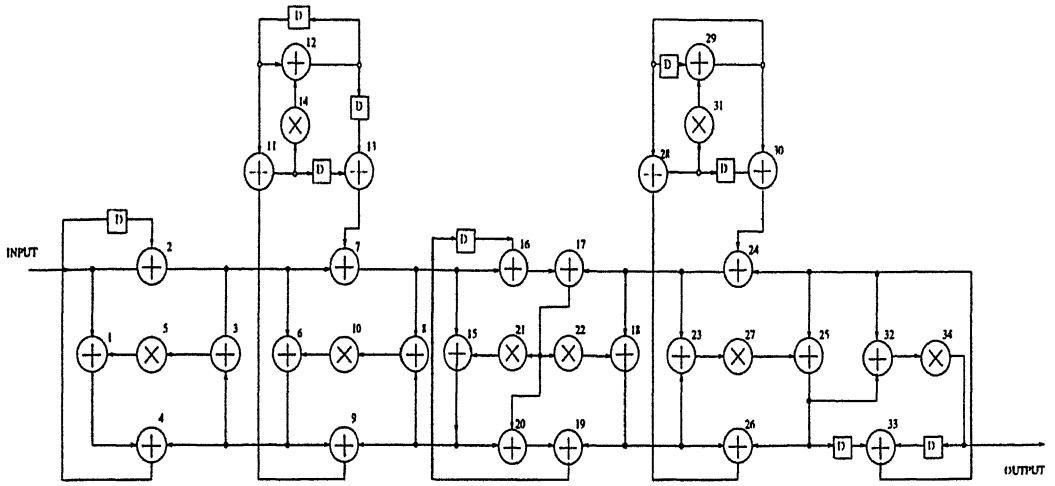


Figure 5.10: Retimed Wave Digital Filter

$$\begin{bmatrix} 1_0 & 4_0 & 2_1 & 7_0 & 16_0 & 17_0 & 22_0 & 22_0 & 18_0 & 23_0 & 27_0 & 27_0 & 25_0 & 32_0 & 34_0 & 34_0 \\ & & 13_1 & 3_1 & 8_0 & 9_0 & 11_0 & 14_0 & 14_0 & 12_0 & & & & 33_0 & 5_1 & 5_1 \\ & & & 6_1 & 15_0 & 10_1 & 10_1 & & & 26_0 & 28_0 & 31_0 & 31_0 & 29_0 & & \\ & & & & & 20_0 & 21_0 & 21_0 & & 19_0 & 30_0 & 24_1 & & & & \end{bmatrix}$$

5.4.2 Resource Allocation

FU Allocation

The number of adders required for this graph is 4 and the number of multipliers required are 3.The nodes binded to various functional units are listed in Table 5.7.

Functional Unit	*Operation Allocated*							
Adder 1	Nodes	1	2	3	4	15	20	23
	Time	0	2	3	1	4	5	9
Adder 2	Nodes	7	8	13	18	19	24	29
	Time	3	5	2	8	9	12	13
Adder 3	Nodes	6	11	16	17	26	33	
	Time	3	6	4	5	9	13	
Adder 4	Nodes	9	12	25	28	30	32	
	Time	5	9	12	10	11	13	
Multiplier 1	Nodes	5	21	27				
	Time	14	6	10				
Multiplier 2	Nodes	10	14	34				
	Time	5	7	14				
Multiplier 3	Nodes	22	31					
	Time	6	11					

Table 5.7: FU - Node Mapping

Register Alloaction

The lifetime chart of various variables in the scheduled graph is shown in the Table 5.8.

node 1	L	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
node 2	D	D	L	D	D	D	D	D	D	D	D	D	D	D	D	D
node 3	D	D	D	L	L	L	L	L	D	D	D	D	D	D	D	D
node 4	D	L	D	D	D	D	D	D	D	D	D	D	D	D	D	D
node 5	D	D	D	D	D	D	D	D	D	L	L	L	L	L	L	L
node 6	L	D	D	L	L	L	L	L	L	L	L	L	L	L	L	L
node 7	D	D	D	L	D	D	D	D	D	D	D	D	D	D	D	D
node 8	D	D	D	D	L	L	L	D	D	D	D	D	D	D	D	D
node 9	D	D	D	D	L	D	D	D	D	D	D	D	D	D	D	D
node 10	L	D	D	D	D	D	D	D	L	L	L	L	L	L	L	L
node 11	D	D	D	D	D	L	L	D	D	D	D	D	D	D	D	D
node 12	L	D	D	D	D	D	D	D	L	L	L	L	L	L	L	L
node 13	L	D	D	D	D	D	D	L	L	L	L	L	L	L	L	L
node 14	D	D	D	D	D	D	D	L	D	D	D	D	D	D	D	D
node 15	L	D	D	D	L	L	L	L	L	L	L	L	L	L	L	L
node 16	D	D	D	D	L	D	D	D	D	D	D	D	D	D	D	D
node 17	D	D	D	D	D	L	D	D	D	D	D	D	D	D	D	D
node 18	D	D	D	D	D	D	D	D	L	L	L	L	L	L	L	L
node 19	L	D	D	D	D	D	D	D	D	L	L	L	L	L	L	L
node 20	D	D	D	D	D	D	L	L	L	D	D	D	D	D	D	D
node 21	L	D	D	D	D	D	D	L	L	L	L	L	L	L	L	L
node 22	D	D	D	D	D	D	D	L	D	D	D	D	D	D	D	D
node 23	L	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
node 24	L	D	D	D	D	D	D	D	D	D	D	D	D	D	D	L
node 25	D	D	D	L	L	L	L	L	L	D	D	D	D	D	D	D
node 26	D	D	D	D	D	D	D	D	D	L	D	D	D	D	D	D
node 27	D	D	L	D	D	D	D	D	D	D	D	D	D	D	D	D
node 28	D	D	D	D	D	D	D	D	D	D	L	L	L	L	D	D
node 29	L	D	D	D	D	D	D	D	D	D	D	D	D	L	L	L
node 30	D	D	D	D	D	D	D	D	D	D	D	D	D	D	L	D
node 31	D	D	D	D	D	D	D	D	D	D	D	D	L	D	D	D
node 32	D	D	D	D	L	D	D	D	D	D	D	D	D	D	D	D
node 33	L	D	D	D	D	L	L	L	L	L	L	L	L	L	L	L
node 34	L	D	D	D	D	D	L	L	L	L	L	L	L	L	L	L

Table 5.8: Lifetime Chart for Wave Digital Filter

The minimum number of registers needed equals 20. The architecture details are shown in the Figure 5.11.

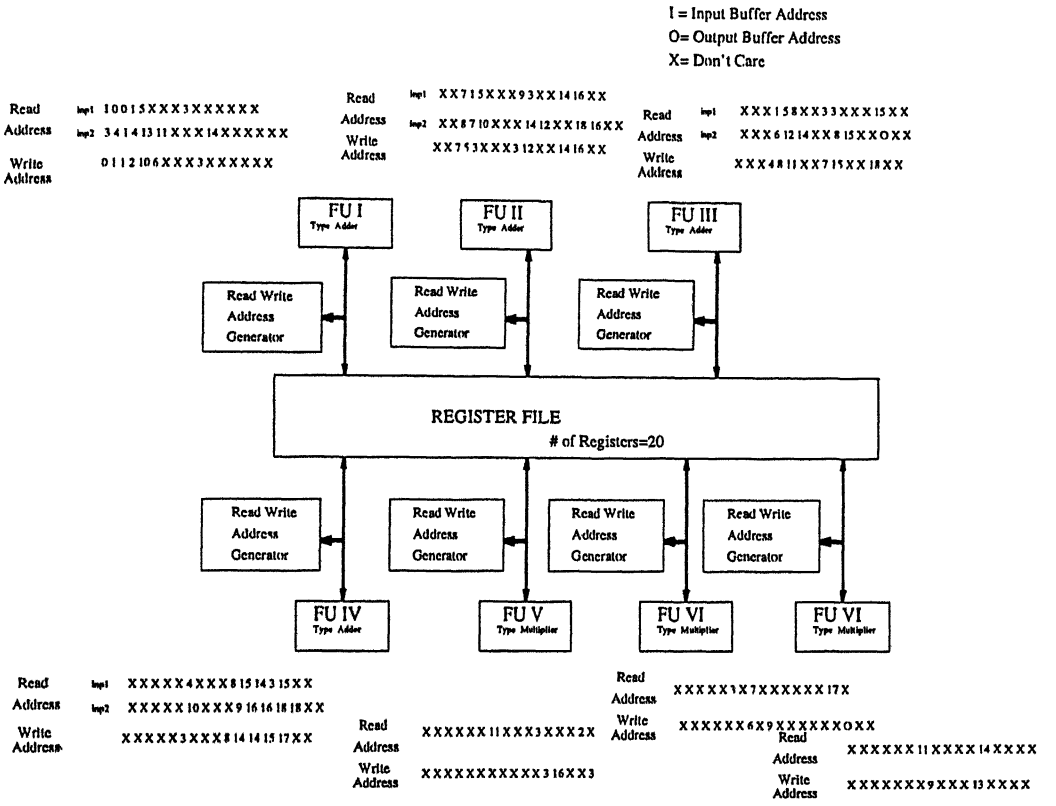


Figure 5.11: Synthesised Architecture for 5th order WDF

5.5 Discussion of Results

In this chapter we have described at length the architectural specifications such as the number of units and registers and control circuits requirements of for various commonly used filter structures. The number of functional units has been reduced to a great extent as compared to the one present in the original DFG. The number of intermediate storage elements required in the synthesized architecture is also less as compared to the number of nodes present in the graph. It may be noted that each node output is assigned to a register

which has been attempted to be minimized. Most of the available designs procedures[1] use complex criss cross connections and use a number of multiplexers and registers. This increases the overall area as the interconnections also occupy some area. It may be noted that in all of the synthesised circuits the number of functional units used is greater than lower bound. However by increasing the number of functional units by unity, the number of registers allocated has been drastically reduced as compared to the available results[1].

For the case of WDF Parhi's[1] methodology has achieved the lower bound on the number of FUs however the synthesised circuit has larger number of registers(> 40) and a complex criss cross connections, with all other conditions remaining the same. Comparing both the designs assuming
Number of gates of an adder = Number of gates of a register = X gates
Number of gates of a multiplier = 4X gates, then
Total number of gates in Parhi's design = $40X + 3X + 8X = 51X$
Total number of gates in obtained = $20X + 4X + 12X = 36X$
Therefore a reduction of 15X gates has been achieved for the case of WDF. Though the decoding circuits of register file takes some area, it is estimated that the number of gates required for multiplexers in Parhi's[1] design and our design the number of gates required for the proposed circuits would be approximately equal due to the simplification of the address sequencing control circuits.

Chapter 6

Conclusions and Scope for Further Work

In this thesis, an attempt has been made towards synthesising some of the basic DSP algorithms. Although the algorithms can be behaviorally described in any of the high level constructs available, for the present work however the algorithm has been represented in a most primitive form i.e. in the form of a graph. As the translation of a high level construct into a graph (dependency relation) is commonly used in the design of compilers etc.[5] [6] and hence, not much attention has been paid towards the same. The attention has been primarily focussed on the scheduling and allocation aspects.The allocation process is targeted towards a proposed architecture. The specifications of different architectures are generated and listed in Chapter 5 for various commonly used signal flow graphs. The process can be easily extended for more complex graphs and for more realistic problems.

For future work more attention can be given towards the automation of design of the various blocks used and implementing of the same on a chip. More research is needed for reducing the number of units further down to the lower

bound. The allocation and scheduling process could be combined, by determining the weights dynamically while scheduling the process. This process may lead to a better design of the overall system. An efficient heuristic needs to be found out for achieving the objective functions listed in Section 4.2 since the present work uses an exhaustive search for achieving the optimal solution.

Appendix

n this section we describe at length the scheduling scheme and hardware allocation for the second order lattice filter shown in Figure 5.1.

Scheduling

From the figure we observe that the data flow graph has three loops. The constituent nodes are listed as:

1. Loop 1: 1 2 6 11 3
2. Loop 2: 1 2 11 3
3. Loop 3: 2 4

Among these loops, Loop 1 and Loop 2 constitute the critical loop since their loop bound is 7 (as explained in Section 3.4). Hence the constituent nodes are marked 'critical'. The number of I/O paths in this flow graph are six. These paths are listed as:

1. Path 1: 1 2 12 13
2. Path 2: 1 2 6 11 7 8 10 13

3. Path 3: 1 2 6 11 9 10 13

4. Path 4: 1 2 11 7 8 10 13

5. Path 5: 1 2 11 9 10 13

6. Patl. 6: 1 5 7 8 10 13

Among all the I/O paths, the path that has the maximum throughput delay is Path 3. Since the throughput delay cannot be compromised, the constituent nodes are also marked as 'critical'. The scheduling algorithm described in Section 3.4 first schedules the critical path and creates the schedule initially as:

$$\begin{bmatrix} 1_0 & 2_0 & 6_0 & 6_0 & 11 & 9_0 & 9_0 \\ 10_{-1} & 13_{-1} & & & & & \end{bmatrix}$$

It may be noted form the matrix that for the zeroth iteration the execution times of nodes 10 and 13 are the time slots 8 and 9 respectively. This is due to the fact that the throughput delay is greater than the iteration bound. After the critical path is scheduled the next crtical node to be selected is node 3. This node is placed in time slot 6 and 7 satisfying the precedence constraints. The next node to be selected is node 4 since the nodes 1,2 & 3 have already been scheduled. This node since has a flexibility, can be placed either after node 2 or before node 2. This node has been placed before node 2 since priority has been given to the slot before that of the scheduled successor. Similarly all other nodes are selected and are placed to get the final schedule matrix described in Section 5.1.

Resource Allocation

From the Figure 5.1 it is clear that the number of different functional units needed in the present case is two. They are multipliers and adders. Both these FUs are allocated different node operations separately. From the schedule matrix is is clear that the minimum number of adders required would be two since there are two such operations which needs to be computed concurrently. The compatibility graph for the adders is as shown in Figure 5.2. As mentioned in Section 4.2 the weight w that is attached to each edge e(u,v,w) between nodes u and v, equals the number of nodes the data computed by u needs to pass through before reaching node v. To form the clique partitions(Section 4.2) it is evident from the graph, that the nodes (1,2) ,(7,11) and (10,13) needs to be combined, since the corresponding edge weights are zero.However the cliques that would finally result would contain the nodes (1,2,7) and (10,13,11). In this way all the three constraints given in Section 4.2 are satisfied. The compatibility graph for multiplier(non pipelined) is also shown in Figure 5.2. in this case the nodes (6,3) (9,12) (4,5,8) gets combined and thus requiring 3 multipliers.

Bibliography

[1] C.Y.Wang and K.K.Parhi, "High-level DSP Synthesis Using Concurrent Transformations, Scheduling, and Allocation," *IEEE Trans. on Computer Aided Design*, vol. 14, pp. 274–295, Mar. 1995.

[2] C.T.Hwang et al, "A Formal Approach to Scheduling Problem in High Level Synthesis," *IEEE Trans. on Computer Aided Design*, vol. 10, pp. 431–447, Apr 1991.

[3] V.K.Madisetti, *VLSI Digital Signal Processing*, IEEE Press, 1995.

[4] K.K.Parhi, "DSP Architectures," *Tutorial,8th International Conference on VLSI Design*, 1995.

[5] H.DeMan et al, "Cathedral II: A Silicon Compiler for Digital Signal Processing," *dt*, vol. 3, pp. 13–25, Dec. 1986.

[6] B.S.Haroun et al, "Architectural Synthesis of DSP Silicon Compiler," *IEEE Trans. on Computer Aided Design*, vol. 10, pp. 431–447, Jun 1990.

[7] D.J.Gajski, *High Level Synthesis*, Kluwer Acacemics, 1992.

[8] P.G.Paulin et al, "Force Directed Scheduling for Behavioral Synthesis of ASIC's," *IEEE Trans. on Computer Aided Design*, vol. 8, pp. 661–679, Jun. 1989.

[9] Narasingh Deo, *Graph Theory with Application to Engineering and Computer Science*, Prentice Hall, 1974.

[10] Michel Gondran & Minoux, *Graph and Algorithms*, John Wiley & sons, 1984.

[11] C.E.Leiserson et al, "Optimizing Synchronous Circuitry by Retiming," *3rd Caltech conf. VLSI*, pp. 87–116, 1983.

[12] K.K.Parhi et al, "Static Rate-Optimal Scheduling of Iterative Data Flow Programs via Optimum Unfolding," *IEEE Trans. on Computer*, vol. 40, pp. 1581–1584, Feb 1991.

[13] V.Vishvanathan, "DSP Architectures," *Tutorial,9th International Conference on VLSI Design*, 1996.

[14] S.Chaudhuri et al, "Computing Lower Bounds on Functional Units before Scheduling," *IEEE Trans. on VLSI Systems*, vol. 4, pp. 273–279, Jun 1996.

[15] R.Composano et al, "The High Level Synthesis of Digital Systems," *Proc. IEEE*, pp. 310–318, Feb. 1990.

.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....

EE-1997-M-ARV-MET